信息工程系

教

案

课程名称： Python 编程基础

教　师： 徐珙

总 学 时 ： 54

理论学时： 18

实训学时： 36

上课班级： 物联网 251，物联网（自主招生）251

授课学期： 2025-2026 学年第二学期

第一章 搭建 Python 开发环境

计划学时：3 学时

一、教学目标

（一）知识目标

1. 了解 Python 语言的特点和运行机制，包括解释型语言的含义以及跨平台特性。
2. 掌握在 Windows 系统下安装 Python 开发环境的步骤，包括下载安装程序、安装过程中的关键选项（如添加到 PATH）以及安装完成后得到的组件（Python 解释器、命令行交互环境、IDLE 和 pip）。
3. 理解 Python 程序的两种运行方式（交互式和文件式），并学会在 IDLE 中使用这两种方式运行简单的“Hello World!”程序。
4. 了解 Visual Studio Code（VS Code）和 Jupyter Notebook 的基本功能及其在 Python 开发中的作用，掌握它们的安装方法以及在其中运行 Python 代码的基本操作。

（二）能力目标

1. 能够独立完成在 Windows 系统下 Python 开发环境的安装，并正确配置环境变量，确保 Python 命令可以在命令行中正常运行。
2. 能够熟练使用 IDLE 进行 Python 代码的编写、保存和运行，包括交互式编程和文件式编程，并能够根据需求选择合适的编程方式。
3. 能够安装并配置 VS Code，安装 Python 扩展，并在 VS Code 中运行 Python 代码，掌握 VS Code 的基本编辑和运行功能。
4. 能够安装 Jupyter Notebook（通过 pip 或 Anaconda），并熟悉其界面和基本操作，包括创建和编辑 Notebook 文件、运行代码单元格、保存 Notebook 等，能够在 Jupyter Notebook 中进行简单的 Python 代码实验和数据分析。
5. 能够根据实际需求选择合适的 Python 开发工具，并在不同的工具之间进行切换和使用，提高 Python 开发的效率和灵活性。

（三）素质目标

1. 培养学生对编程的兴趣和热情，激发学生主动学习和探索 Python 语言及相关开发工具的欲望，为后续深入学习 Python 编程打下良好的基础。
2. 培养学生严谨的逻辑思维能力和问题解决能力，在安装和配置开发环境过程

中遇到问题时，能够冷静思考、分析原因并尝试解决，提高学生面对技术问题时的应对能力。

3. 培养学生自主学习能力和信息获取能力，鼓励学生在课后主动查阅相关资料，了解 Python 开发环境的更多高级配置和优化方法，以及 VS Code 和 Jupyter Notebook 的更多高级功能，拓宽学生的知识面和技术视野。
4. 培养学生的动手实践能力和创新意识，通过在不同的开发工具中运行 Python 代码，让学生在实践中感受编程的乐趣和成就感，鼓励学生尝试使用不同的方法和工具来实现相同的功能，培养学生的创新思维和实践能力。

（四）思政目标

1. 在讲解 Python 的跨平台特性时，引导学生树立开放包容的观念，理解不同操作系统之间的差异和共性，培养学生的多元文化意识和适应能力，使学生明白在多元化的技术环境中，学会适应和利用各种资源是非常重要的。
2. 在安装和配置开发环境的过程中，强调软件的正版使用和合法获取的重要性，培养学生的版权意识和法律意识，让学生明白尊重知识产权是每个公民应尽的义务，也是推动技术发展和社会进步的重要保障。
3. 通过介绍 Python 在各个领域的广泛应用，如数据分析、人工智能、Web 开发等，激发学生的爱国热情和民族自豪感，让学生了解我国在信息技术领域的发展成就，同时也让学生认识到作为新时代的大学生，肩负着推动我国信息技术发展和创新的历史使命，鼓励学生努力学习，为国家的科技进步贡献力量。

二、教学重难点

（一）重点

1. Windows 系统下 Python 开发环境的安装步骤，特别是添加到 PATH 环境变量的重要性以及安装过程中默认组件（IDLE 和 pip）的作用。
2. Python 程序的两种运行方式（交互式和文件式）在 IDLE 中的具体操作方法，包括如何编写、保存和运行代码，以及运行结果的查看和分析。
3. VS Code 和 Jupyter Notebook 的安装方法，以及在这些工具中运行 Python 代码的基本操作流程，如 VS Code 中 Python 扩展的安装和代码运行，Jupyter Notebook 的界面组成、单元格操作和代码运行等。

（二）难点

1. 在安装 Python 开发环境时，如何根据不同的 Windows 系统版本（如 32 位和 64 位）选择合适的安装程序，以及如何处理安装过程中可能出现的兼容性问题和错误提示，确保安装过程顺利进行。
2. 如何让学生理解 Python 作为解释型语言的运行机制，以及这种运行机制与编译型语言的区别，通过具体的实例和对比分析，帮助学生深入理解解释器的作用和 Python 代码的执行过程。
3. Jupyter Notebook 的高级功能和配置，如修改默认浏览器和默认工作路径的设置方法，以及如何在 VS Code 中集成和使用 Jupyter Notebook，这些内容对于初学者来说可能会有一定的难度，需要通过详细的讲解和演示，让学生逐步掌握这些高级操作技巧。

三、教学方法

1. **讲授法**：通过系统的讲解，向学生介绍 Python 语言的特点、运行机制、开发环境的安装步骤以及各种开发工具的基本功能和使用方法，帮助学生建立起完整的知识体系，使学生对 Python 开发环境有一个清晰的认识和理解。
2. **演示法**：在讲解开发环境的安装和配置过程中，通过实际操作演示，让学生直观地看到每个步骤的具体操作方法和效果，如 Python 安装程序的下载和安装过程、IDLE 的使用、VS Code 和 Jupyter Notebook 的安装与配置等，同时在演示过程中详细讲解每个操作步骤的目的和注意事项，帮助学生更好地理解和掌握操作要点。
3. **实践操作法**：安排学生在课堂上进行实际操作练习，让学生在亲身体验中掌握 Python 开发环境的搭建和使用方法。例如，让学生自己下载并安装 Python 开发环境，尝试在 IDLE 中运行“Hello World!”程序，安装并配置 VS Code 和 Jupyter Notebook 等，通过实践操作，学生可以加深对知识的理解，提高动手能力和解决实际问题的能力。

四、教学过程

（一）课前思政、素质元素导入

在当今数字化时代，编程语言作为信息技术的核心工具，对于推动社会进步和经济发展具有重要意义。Python 语言以其简洁易学、功能强大的特点，在全球

范围内得到了广泛应用。通过学习 Python 开发环境的搭建，我们不仅能够掌握一门实用的编程技术，还能够培养自己的逻辑思维能力和问题解决能力。同时，我们也要树立正确的价值观，尊重知识产权，合法使用软件资源，为营造良好的技术发展环境贡献自己的力量。此外，Python 语言的跨平台特性也体现了技术的包容性和开放性，我们要学会在不同的操作系统和环境中灵活运用所学知识，培养自己的适应能力和创新精神。

（二）教学内容

1. **Python 语言概述**

- **定义与特点**：介绍 Python 是一种解释型的高级程序设计语言，解释器的作用是将源代码逐条转换成目标代码并运行。强调 Python 的跨平台特性，可以在 Windows、Mac 和各种 Linux/Unix 操作系统上运行，这使得 Python 在全球范围内得到了广泛应用。
- **应用场景**：简要介绍 Python 在数据分析、人工智能、Web 开发、自动化脚本等领域的广泛应用，让学生了解学习 Python 的重要性和实用性，激发学生的学习兴趣。

2. **Windows 系统下 Python 开发环境的安装**

- **下载 Python 安装程序**：指导学生访问 Python 官网。讲解如何根据自己的 Windows 系统版本（32 位或 64 位）选择合适的安装程序，同时提醒学生注意 Python 3.9 以后的版本无法安装到 Windows 7 及之前的系统。
- **安装 Python**：详细演示 Python 安装程序的安装过程。首先，双击下载好的安装程序，进入安装界面后，强调要勾选“Add Python 3.x to PATH”选项，以便将 Python 路径添加到系统环境变量 PATH 中，方便后续在命令行中运行 Python 命令。然后点击“Install Now”开始安装，默认安装路径即可。介绍安装过程中 Python 会自动安装 IDLE 和 pip，分别说明 IDLE 是 Python 自带的集成开发环境，pip 是 Python 包管理工具，它们在后续 Python 开发中的重要作用。
- **验证安装**：安装完成后，通过在命令行中输入“python”或“python -v”命令，检查 Python 是否安装成功，查看 Python 的版本信息。同时，输入“pip -v”命令，验证 pip 是否正常工作，让学生学会如何检查开发环境的安装状

态，确保环境搭建成功。

3. **运行第一个 Python 程序——“Hello World!”**

- **交互式运行**：在开始菜单中选择“Python 3.x”->“IDLE”打开 IDLE，启动交互式窗口。在提示符“>>>”后输入代码 print("Hello World!")，讲解 print()是 Python 的输出函数，用于在屏幕上输出内容。按 Enter 键运行代码，让学生观察运行结果，屏幕上输出“Hello World!”。通过这个简单的交互式编程示例，让学生初步感受 Python 代码的编写和运行过程，了解交互式编程的特点。
- **文件式运行**:进入 IDLE 的文件式编程环境,通过选择“File”->“New File”或使用快捷键“Ctrl + N”打开 Python 程序编辑窗口。在编辑窗口中输入代码 print("Hello World!")，然后通过选择“File”->“Save”/“Save As...”或使用快捷键“Ctrl + S”将文件保存为扩展名为“.py”的 Python 文件，例如保存为“hello_world.py”。接着，在程序编辑窗口的菜单上选择“Run”->“Run Module”或使用快捷键 F5 运行代码，让学生观察运行结果输出在 IDLE 窗口。讲解文件式编程与交互式编程的区别，以及文件式编程在实际项目开发中的优势，如便于代码的保存、管理和复用。

4. **安装和使用 Visual Studio Code（VS Code）**

- **安装 VS Code**：介绍 VS Code 是一款轻量级但功能强大的代码编辑器，支持多种编程语言，包括 Python。指导学生访问 VS Code 官网，根据自己的操作系统版本下载合适的安装程序。特别提醒学生，VS Code 支持 Windows 7 的最后版本为 17.0.3，如果使用 Windows 7 系统，需要下载对应版本。下载完成后，双击安装程序，进入安装界面，注意要勾选所有选项，完成 VS Code 的安装。
- **在 VS Code 中使用 Python**：首先，强调在 VS Code 中运行 Python 代码之前，需要确保已经安装了 Python 解释器。然后，在 VS Code 左侧的侧边栏点击“Extensions（扩展）”，在搜索框里输入“Python”，找到 Python 扩展后点击“Install（安装）”按钮进行安装。安装完成后，在 VS Code 菜单栏选择“File（文件）”->“New Text File（新建文本文件）”，或使用快捷键 Ctrl+N 新建一个文本文件。在新建的文本文件上方，点击“Select a

language（选择语言）”，在下拉列表中选择“Python”。接着，在文本文件中输入代码 print("Hello World!")，按快捷键 Ctrl+S 保存文件，文件扩展名为“.py”。最后，通过点击 VS Code 编辑器文件右上角的三角形播放按钮，或者在文件任意位置右击选择“Run Python”->“Run Python File in Terminal”，运行 Python 代码，并让学生观察运行结果在终端面板中的显示。通过这个过程，让学生熟悉在 VS Code 中进行 Python 开发的基本操作流程，了解 VS Code 相比 IDLE 在代码编辑和运行方面的一些优势，如更强大的代码提示、调试功能等。

5. **安装和使用 Jupyter Notebook**

- **安装 Jupyter Notebook：**
  - **方法一：使用 pip 安装：**首先，回顾 pip 的作用，它是 Python 的包管理工具，用于安装和管理 Python 包。指导学生按 Win+R 快捷键，在弹出的窗口输入“cmd”，打开命令提示符窗口。在命令提示符窗口中，输入命令 pip install jupyter notebook，开始下载和安装 Jupyter Notebook。如果网络连接较差，可以使用清华镜像站来下载和安装，命令是 pip install -i https://pypi.tuna.tsinghua.edu.cn/simple jupyter notebook。安装完成后，在命令行窗口输入“pip list”，让学生观察列表中是否出现了“jupyter”，以验证 Jupyter Notebook 是否安装成功。然后，输入“jupyter notebook”命令启动 Jupyter Notebook，让学生观察 Jupyter Notebook 在默认网页浏览器中打开的新标签页面。
  - **方法二：安装 Anaconda：**介绍 Anaconda 是一个集成了很多科学计算需要使用的 Python 第三方工具包的 Python IDE，并且预装了 Jupyter Notebook。推荐学生在国内下载 Anaconda 时，可以通过清华大学开源软件镜像站。启动 Jupyter Notebook，同样会在默认网页浏览器中打开一个新的标签。
- **使用 Jupyter Notebook：**启动 Jupyter Notebook 后，让学生熟悉其主面板的布局，包括 Files、Running、Clusters 等选项卡。在 Files 选项卡下，右上方点击“New”下拉按钮，选择“Notebook”->“Python 3”，新建一个 Notebook（扩展名为“.ipynb”），并进入新 Notebook 的编辑界面。介绍编辑界面的主要组成部分，包括名称、菜单栏、工具条以及单元格（Cell）。

讲解单元格的两种格式：代码（Code）单元格和注释说明（Markdown）单元格，以及如何通过菜单上的“Cell”->“Cell Type”改变单元格的类型。演示如何在代码单元格中输入代码 print("Hello World!")，按下“Ctrl + Enter”执行选中的单元格，让学生观察执行结果显示在代码单元格下方。同时，介绍“Shift + Enter”的作用，即执行选中的单元格，并选中下一个单元格(如果执行的是最后一个单元格,会在下方插入一个新的单元格并选中)。最后，讲解如何保存 Notebook，包括点击工具条上的保存按钮以及菜单栏上的“File”里的更多保存选项。

- **在 VS Code 中使用 Jupyter Notebook**：介绍 VS Code 对 Jupyter Notebook 的本地支持。首先,确保已经按照前面的步骤安装好了 Python 环境和 Jupyter Notebook 包。然后，在 VS Code 中，通过在左侧的侧边栏点击“Extensions（扩展）”，在搜索框里输入“Jupyter”，找到 Jupyter 扩展后点击“Install（安装）”按钮进行安装。安装完成后，在 VS Code 中打开已有 Jupyter Notebook 文件，或通过选择菜单栏中的“File（文件）”->“New File（新建文件）”，在弹出的下拉列表中选择“Jupyter Notebook”来新建一个 Jupyter Notebook 文件。让学生熟悉在 VS Code 中创建和编辑 Jupyter Notebook 文件的操作，包括在编辑区上方按“+ Code”按钮创建新的代码单元格，按“+ Markdown”按钮创建新的 Markdown 单元格，以及每个代码单元格左边的三角形运行按钮的使用方法。此外，介绍单元格右上角的“...”按钮提供的单元格操作功能，如剪切、复制、粘贴单元格，在选中单元格上方或下方插入代码单元格或 Markdown 单元格等，让学生体验在 VS Code 中使用 Jupyter Notebook 的便捷性和灵活性。

（三）课堂总结

1. 回顾 Python 语言的特点和运行机制，强调其作为解释型语言的优势以及跨平台特性的重要性。
2. 总结 Windows 系统下 Python 开发环境的安装步骤，包括下载安装程序、安装过程中的关键选项以及安装完成后得到的组件，提醒学生在安装过程中要注意的细节和常见问题的解决方法。
3. 对比 Python 程序的交互式和文件式运行方式，让学生明确它们的适用场景和

优缺点，加深对这两种编程方式的理解。

4. 梳理 VS Code 和 Jupyter Notebook 的安装方法以及在它们中运行 Python 代码的基本操作流程，强调这些开发工具在 Python 开发中的作用和优势，鼓励学生在课后进一步探索这些工具的高级功能和使用技巧。
5. 强调在学习 Python 开发过程中，要注重培养自己的动手实践能力和问题解决能力，遇到问题时要善于思考和尝试解决，同时要保持对新技术的好奇心和学习热情，不断提升自己的编程水平。

（四）布置作业

1. **理论作业：**

- 请简述 Python 语言的特点和运行机制，并解释为什么 Python 是一种跨平台的语言。
- 对比 Python 程序的交互式运行方式和文件式运行方式，分别列举它们的优点和适用场景。

2. **实践作业：**

- 在自己的计算机上安装 Python 开发环境，并确保安装成功。
- 使用 IDLE 编写一个 Python 程序。要求分别用交互式和文件式两种方式运行该程序，并将运行结果截图保存。
- 安装 Visual Studio Code 和 Jupyter Notebook（可选择其中一种安装方式），并在它们中分别运行一个简单的 Python 程序，如输出自己的姓名和学号。将安装和运行过程中的关键步骤截图。

第二章 Python 基础语法

计划学时：3 学时

一、教学目标

（一）知识目标

1. 理解 Python 中行与缩进的规则，包括如何使用反斜杠进行续行以及缩进的规范。
2. 掌握 Python 中单行和多行注释的使用方法。
3. 熟悉 Python 标识符和关键字的定义及使用规则。
4. 了解 Python 的六种标准数据类型（Number、String、List、Tuple、Set、Dictionary）及其特点。
5. 掌握变量的定义和使用方法，包括变量赋值、数据类型判断。
6. 理解 Python 中不可变数据类型和可变数据类型的区别。
7. 掌握 print()函数和 input()函数的使用方法，包括它们的参数设置及作用。

（二）能力目标

1. 能够规范地书写 Python 代码，合理使用缩进和续行，提高代码可读性。
2. 能够熟练使用注释对代码进行说明，便于代码维护和理解。
3. 能够正确使用标识符命名变量和函数，避免与关键字冲突。
4. 能够根据需求选择合适的数据类型，并正确使用变量进行数据存储和操作。
5. 能够灵活运用 print()函数实现数据的格式化输出，以及使用 input()函数实现数据的输入。

（三）素质目标

1. 培养学生严谨的编程习惯，注重代码的规范性和可读性。
2. 提高学生分析问题和解决问题的能力，通过实践操作加深对 Python 基础语法的理解。
3. 激发学生对 Python 编程的兴趣，培养学生的自主学习能力和探索精神。

（四）思政目标

1. 引导学生树立正确的编程价值观，注重代码质量，培养良好的编程职业道德。

2. 通过学习 Python 基础语法，让学生感受到编程语言的严谨性和逻辑性，培养学生严谨的思维方式和科学态度。

二、教学重难点

（一）重点

1. Python 中行与缩进的规则，特别是缩进的规范和续行的使用方法。
2. 注释的使用方法，包括单行注释和多行注释的格式及作用。
3. 标识符和关键字的定义及使用规则，以及如何避免与关键字冲突。
4. 六种标准数据类型的定义及特点，以及变量的定义和使用方法。
5. print()函数和 input()函数的使用方法，包括它们的参数设置及作用。

（二）难点

1. 理解 Python 中不可变数据类型和可变数据类型的区别，以及它们在实际编程中的应用。
2. 掌握 print()函数中参数 sep 和 end 的灵活使用,实现数据的格式化输出。

三、教学方法

1. **讲授法**：通过系统的讲解，向学生介绍 Python 基础语法的相关知识，包括行与缩进、注释、标识符和关键字、数据类型、变量的定义和使用、print()函数和 input()函数等，帮助学生建立起完整的知识体系。
2. **演示法**：通过实际操作演示，让学生直观地看到 Python 代码的编写和运行过程，如如何使用缩进和续行、如何添加注释、如何定义变量和使用数据类型、如何使用 print()函数和 input()函数等，同时在演示过程中详细讲解每个操作步骤的目的和注意事项,帮助学生更好地理解和掌握操作要点。
3. **实践操作法**：安排学生在课堂上进行实际操作练习，让学生在亲身体验中掌握 Python 基础语法的使用方法。例如，让学生自己编写带有缩进和续行的代码、添加注释、定义变量、使用 print()函数和 input()函数等，通过实践操作，学生可以加深对知识的理解，提高动手能力和解决实际问题的能力。

4. **案例分析法**：通过分析一些简单的 Python 编程案例，如数据输入输出案例、变量使用案例等，让学生了解 Python 基础语法在实际编程中的应用，帮助学生将理论知识与实际应用相结合，提高学生对知识的综合运用能力。

四、教学过程

（一）课前思政、素质元素导入

在当今数字化时代，编程已经成为一种重要的技能，而 Python 作为一种简洁易学且功能强大的编程语言，受到了广泛的应用。学习 Python 基础语法不仅是掌握一门编程语言的开始，更是培养严谨编程习惯和逻辑思维能力的重要过程。在编程过程中，我们要注重代码的规范性和可读性，这不仅有助于我们自己更好地理解和维护代码，也方便他人阅读和协作。同时，良好的编程习惯也是一种职业素养的体现，希望大家在学习过程中能够树立正确的编程价值观，注重代码质量，培养良好的编程职业道德。

（二）教学内容

1. **行与缩进**
    - **行的书写规则**：介绍 Python 通常是一行写一条语句，如果一行内写多条语句，语句之间应使用分号分隔，但建议每行只写一条语句，语句结束时不使用分号。通过示例代码展示这种书写方式。
    - **续行方法**：讲解如果一条语句太长，可以使用反斜杠\来进行续行，并给出示例代码，让学生观察续行的效果。
    - **缩进规则**：强调 Python 的一个特色是使用缩进来表示代码块，缩进的空格数没有规定，但同一个代码块的语句必须包含相同的缩进空格数。推荐每个缩进级别使用 4 个空格，不推荐使用 Tab。通过示例代码展示正确的缩进方式以及错误的缩进方式可能导致的错误。
2. **注释**
    - **注释的作用**：讲解注释在程序设计中的重要性，它不会被解释和运行，主要目的是便于对程序进行维护。
    - **单行注释**：介绍 Python 中单行注释以#开头，并给出示例代码，展示单行注释的使用方法。

- **多行注释**：讲解多行注释可以用多个#号，每行一个，也可以使用三引号（'''或"""）。通过示例代码展示多行注释的使用方法。

3. **标识符和关键字**
   - **标识符的定义及规则**：说明 Python 使用不同的标识符来标识变量名、函数名等，标识符是一个字符序列，第一个字符必须是字母表中字母或下划线_，其他部分由字母、数字和下划线组成，标识符对大小写敏感。给出几个合法的标识符示例。
   - **关键字**：强调 Python 语言保留了某些单词用作特殊用途，这些单词称为保留字或关键字，用户定义的标识符不能和关键字相同。介绍 Python 的标准库提供了一个 keyword 模块，可以输出当前版本的所有关键字，并通过代码示例展示如何使用 keyword.kwlist 输出所有关键字。
4. **变量和数据类型**
   - **数据类型分类**：介绍 Python 3 中的六个标准数据类型：Number（数字）、String（字符串）、List（列表）、Tuple（元组）、Set（集合）、Dictionary（字典）。
   - **变量的定义和使用**：讲解在 Python 中变量不需要提前声明，创建时直接赋值即可，变量的数据类型是由赋值给变量的值决定的。赋值运算符=用于给变量赋值，运算符左边为变量名，右边为变量的值。通过代码示例展示变量的定义和使用方法。
   - **数据类型判断**：介绍 type()函数用于返回变量的类型，isinstance()函数用于判断一个对象是否为一个已知的类型，并通过代码示例展示这两个函数的使用方法。
   - **不可变数据类型与可变数据类型**：解释不可变数据类型和可变数据类型的定义及区别，不可变数据类型包括数字类型、字符串和元组，可变数据类型包括列表、字典、集合。通过示例代码展示不可变数据类型和可变数据类型在值改变时内存地址的变化情况。
5. **数据的输入和输出**
   - **print()函数**

- **函数作用**：介绍 print()函数用于输出数据，默认输出到标准输出设备（显示器）。
- **语法格式**：讲解 print()函数的语法格式 print(*args, sep=' ', end='\n')，其中 args 为输出的对象，输出多个对象时需要用逗号分隔；sep 参数用于设定输出多个对象时的间隔，默认值是一个空格；end 参数用于设定输出的结尾，默认值是换行符\n。
- **参数使用示例**：通过多个代码示例展示 print()函数中 sep 和 end 参数的灵活使用，如实现不换行输出、设置不同的间隔符号等。

- **input()函数**
  - **函数作用**：介绍 input()函数用于从标准输入设备（键盘）读入一行文本，该函数返回一个字符串类型数据。
  - **语法格式**：讲解 input()函数的语法格式 input(prompt='')，其中 prompt 为提示字符串，运行时原样显示，给用户进行提示，默认为空字符串。
  - **使用示例**：通过代码示例展示如何使用 input()函数提示用户输入数据，并将输入内容输出。

（三）课堂总结

1. 回顾 Python 中行与缩进的规则，强调缩进的规范和续行的使用方法，让学生明白正确的缩进和续行对于代码可读性的重要性。
2. 总结注释的使用方法，包括单行注释和多行注释的格式及作用，提醒学生在编程过程中要养成良好的注释习惯，便于代码的维护和理解。
3. 梳理标识符和关键字的定义及使用规则，让学生清楚标识符的命名规范以及关键字的特殊性，避免在编程中出现与关键字冲突的情况。
4. 复习六种标准数据类型及其特点，以及变量的定义和使用方法，让学生掌握如何根据需求选择合适的数据类型，并正确使用变量进行数据存储和操作。

5. 强调 print()函数和 input()函数的使用方法，特别是 print()函数中 sep 和 end 参数的灵活使用，以及 input()函数的输入输出功能，让学生能够熟练运用这两个函数实现数据的输入和格式化输出。

（四）布置作业

第三章 数字类型

计划学时：6 学时

一、教学目标

（一）知识目标

1. 理解数字类型的概念，包括整型（int）、浮点型（float）、复数类型（complex）、布尔类型（bool）。
2. 掌握数字类型的特点，特别是数字类型是不可变数据类型。
3. 熟悉整型的表示方法，包括二进制、八进制、十进制和十六进制。
4. 了解浮点型的表示方法，包括十进制小数形式和科学记数法。
5. 理解复数类型的一般形式及创建方法。
6. 掌握布尔类型的取值及转换规则。
7. 熟悉 Python 中的类型转换方法，包括 int()、float()、bool()等函数的使用。
8. 掌握 Python 中的运算符，包括算术运算符、比较运算符、赋值运算符、逻辑运算符及运算符优先级。
9. 了解常用数学函数，如 abs()、round()、math.ceil()、math.floor()、math.exp()、math.log()、math.sin()等。

（二）能力目标

1. 能够正确使用数字类型进行变量定义和赋值。
2. 能够熟练进行不同进制的整型表示及转换。
3. 能够正确使用浮点型和复数类型进行数学运算。
4. 能够根据需要进行类型转换，将字符串转换为数字类型，或将数字类型转换为布尔类型等。
5. 能够运用各种运算符进行数学运算和逻辑判断。
6. 能够使用常用数学函数进行数学计算。

（三）素质目标

1. 培养学生严谨的逻辑思维能力，通过数字类型的学习和运算符的使用，提高学生分析和解决问题的能力。

2. 提高学生的自主学习能力，引导学生在学习过程中善于思考、勇于探索，培养学生的创新精神。
3. 培养学生良好的编程习惯，注重代码的规范性和可读性，提高学生的编程能力。

（四）思政目标

1. 引导学生树立正确的学习态度，注重基础知识的学习和积累，培养学生的科学精神和严谨态度。
2. 通过学习数字类型和运算符，让学生感受到数学知识在编程中的重要性，激发学生对数学的兴趣和热爱，培养学生的数学素养。

二、教学重难点

（一）重点

1. 数字类型的概念及特点，特别是数字类型是不可变数据类型。
2. 整型的表示方法，包括二进制、八进制、十进制和十六进制。
3. 浮点型的表示方法，包括十进制小数形式和科学记数法。
4. 复数类型的一般形式及创建方法。
5. 布尔类型的取值及转换规则。
6. 类型转换方法，包括 int()、float()、bool()等函数的使用。
7. 运算符的使用方法，包括算术运算符、比较运算符、赋值运算符、逻辑运算符及运算符优先级。
8. 常用数学函数的使用方法，如 abs()、round()、math.ceil()、math.floor()、math.exp()、math.log()、math.sin()等。

（二）难点

1. 理解复数类型的一般形式及创建方法，特别是通过内置函数 complex()创建复数。
2. 掌握类型转换的规则，特别是将字符串转换为数字类型时的格式要求，以及 int()函数将浮点数转换为整数时的截断规则。
3. 理解运算符优先级，特别是逻辑运算符与其他运算符之间的优先级关系。
4. 掌握常用数学函数的使用方法，特别是 math 模块中函数的使用，如 math.ceil()、math.floor()、math.exp()、math.log()、math.sin()等。

三、教学方法

1. **讲授法：**通过系统的讲解，向学生介绍 Python 数字类型的相关知识，包括数字类型的概念、特点、表示方法、类型转换、运算符及常用数学函数等，帮助学生建立起完整的知识体系。
2. **演示法：**通过实际操作演示，让学生直观地看到数字类型的使用方法、类型转换的过程、运算符的运算结果以及常用数学函数的使用效果，同时在演示过程中详细讲解每个操作步骤的目的和注意事项，帮助学生更好地理解和掌握操作要点。
3. **实践操作法：**安排学生在课堂上进行实际操作练习，让学生在亲身体验中掌握数字类型的使用方法、类型转换、运算符的使用以及常用数学函数的应用。例如，让学生自己定义不同类型的数字变量、进行类型转换、使用运算符进行数学运算和逻辑判断、调用常用数学函数进行数学计算等，通过实践操作，学生可以加深对知识的理解，提高动手能力和解决实际问题的能力。

四、教学过程

（一）课前思政、素质元素导入

在当今数字化时代，编程已经成为一种重要的技能，而 Python 作为一种简洁易学且功能强大的编程语言，受到了广泛的应用。数字类型是 Python 中最基本的数据类型之一，它在编程中扮演着重要的角色。通过学习数字类型，不仅可以让我们更好地理解和使用 Python 语言，还可以培养我们的逻辑思维能力和数学素养。在学习过程中，我们要注重基础知识的学习和积累，树立正确的学习态度，培养科学精神和严谨态度，希望大家能够认真对待这节课的学习内容。

（二）教学内容

1. **数字类型概述**
    - **定义：**介绍数字类型是用于表示数字或数值的数据类型，包括整型（int）、浮点型（float）、复数类型（complex）、布尔类型（bool）。
    - **特点：**强调数字类型是不可变的数据类型，如果改变数字类型的变量的值，将重新分配内存空间。

2. **整型（int）**
    - **定义**：说明整型数据即整数，不带小数点，可以有正号或者负号。Python 3 整型是没有限制大小的。
    - **表示方法**：介绍 Python 中整型常量可以用二进制、八进制、十进制和十六进制 4 种形式表示，分别以 0b、0o、无前缀、0x 为前缀。通过代码示例展示不同进制的整型表示方法：

```
num_1 = 100  # 十进制正整数
num_2 = -80  # 十进制负整数
num_3 = 0b1010  # 二进制整数
num_4 = 0o12  # 八进制整数
num_5 = 0x2A  # 十六进制整数
```

3. **浮点型（float）**
    - **定义**：介绍浮点数一般以十进制小数的形式表示。对于较大或较小的浮点数，可以使用科学记数法表示（用字母 e 或 E 表示以 10 为底的指数）。
    - **表示方法**：通过代码示例展示浮点型的表示方法：

```
num_6 = 3.14  # 十进制小数形式
num_7 = -2.3e3  # 科学记数法
num_8 = 2.3E-3  # 科学记数法
```

4. **复数类型（complex）**
    - **定义**：说明复数由实部和虚部构成，其一般形式为 real + imag * 1j。实部 real 和虚部 imag 都是浮点型。
    - **创建方法**：介绍 Python 中有两种创建复数的方式，一种是按照复数的一般形式直接创建，另一种是通过内置函数 complex()创建。通过代码示例展示复数的创建方法：

```
num_9 = 3.5 + 2j  # 按照复数的一般形式直接创建
num_10 = complex(4, -5.4)  # 通过内置函数 complex()创建
```

5. **布尔类型（bool）**

- **定义**：说明布尔类型是整型的子类型，只有两个取值：True 和 False，分别表示逻辑真和逻辑假。
- **转换规则**：介绍 Python 中的任何对象都可以用 bool()函数转换为布尔类型。规则如下：
    - 符合以下条件的数据都会被转换为 False：
        - None；
        - 任何为 0 的数字类型，如 0、0.0、0j；
        - 任何空序列，如""、()、[]；
        - 任何空字典、空集合，如{}；
        - 用户定义的类实例，如类中定义了__bool__()或者__len__()。
    - 除以上对象外，其它的对象都会被转换为 True。
- **代码示例**：

```
print(bool(-10))  # True
print(bool(""))  # False
```

6. **类型转换**
    - **定义**：介绍 Python 内置了一系列可实现强制（或称为显式）类型转换的函数，将数据转换为指定的类型。只需要将数据类型作为函数名即可。
    - **常用类型转换函数**：
        - int(x)：将 x 转换为一个整数。注意：如果要把字符串转换为 int，int()函数只能转换符合数字类型格式规范的字符串。使用 int()函数将浮点数转换为整数时，若有必要会发生截断（取整）而非四舍五入。
        - float(x)：将 x 转换为一个浮点数。注意：如果要把字符串转换为 float，float()函数只能转换符合数字类型格式规范的字符串。
        - bool(x)：将 x 转换为布尔类型。
    - **代码示例**：

```
a = float(1)  # 把整数 1 转换为浮点数
print(a)  # 输出：1.0
b = float("2.89")  # 把字符串转为浮点数 2.89
c = int(b)  # 把浮点数转为整数，发生截断，结果为 2
print(c)  # 输出：2
p = input("请输入一个整数：")  # input()函数从键盘输入，返回的是字符串类型的数据
print(type(p))  # 输出：<class 'str'>
q = int(input("请输入一个整数："))  # 把从键盘输入的字符串转换为整数类型
print(type(q))  # 输出：<class 'int'>
```

7. **运算符**
    - **算术运算符**
        - **定义**：介绍算术运算符包括+、-、*、/、//、%、**，这些运算符都是双目运算符，每个运算符可以与两个操作数组成一个表达式。
        - **运算符说明：**
            - +运算符：加法，x+y 表示求 x 与 y 的和。
            - -运算符：减法，x-y 表示求 x 与 y 的差。
            - *运算符：乘法，x*y 表示求 x 与 y 的积。
            - /运算符：除法，x/y 表示求 x 与 y 的商。
            - //运算符：取整除，x//y 表示求不大于 x 与 y 之商的最大整数。
            - %运算符：取模，x%y 表示求 x 除以 y 的余数。
            - **运算符：幂运算，x**y 表示求 x 的 y 次方。
        - **代码示例：**

```
x = 3
y = 4
print("x + y =", x + y)  # 输出：x + y = 7
```

```
print("x - y =", x - y)  # 输出：x - y = -1
print("x * y =", x * y)  # 输出：x * y = 12
print("x / y =", x / y)  # 输出：x / y = 0.75
print("x // y =", x // y)  # 输出：x // y = 0
print("x % y =", x % y)  # 输出：x % y = 3
print("x ** y =", x ** y)  # 输出：x ** y = 81
```

    - **类型转换规则**：介绍 Python 中的算术运算符支持对相同或不同类型的数字进行混合运算。Python 在对不同类型的对象进行运算时，会强制将对象的类型进行临时类型转换，这些转换遵循如下规则：
        - 布尔类型进行算术运算时，被视为数值 0 或 1。
        - 整型与浮点型运算时，将整型转化为浮点型。
        - 其它类型与复数运算时，将其它类型转换为复数类型。
    - **代码示例**：

```
print(3 + 4.5)  # 输出：7.5
print(True + 2)  # 输出：3
```

- **比较运算符**
    - **定义**：介绍比较运算符有==、!=、>、<、>=、<=。比较运算符同样是双目运算符，它与两个操作数构成一个表达式。若比较结果符合给定的条件，则结果为 True，否则结果就是 False。
    - **运算符说明**：
        - ==运算符：比较左值和右值，若相同则为 True，否则为 False。
        - !=运算符：比较左值和右值，若不相同则为 True，否则为 False。
        - >运算符：比较左值和右值，若左值大于右值则为 True，否则为 False。

        - <运算符：比较左值和右值，若左值小于右值则为True，否则为False。
        - >=运算符：比较左值和右值，若左值大于或等于右值则为True，否则为False。
        - <=运算符：比较左值和右值，若左值小于或等于右值则为True，否则为False。
    - **代码示例：**

```
print(1 == 2)  # 输出：False
print(5 >= 3)  # 输出：True
```

- **赋值运算符**
    - **定义**：介绍赋值运算符的功能是将一个表达式或对象赋给一个左值，其中左值必须是一个可修改的值，不能为一个常量。
    - **基本赋值运算符**：=是基本的赋值运算符。
    - **复合赋值运算符**：介绍=可与算术运算符组合成复合赋值运算符。复合赋值运算符有+=、-=、*=、/=、//=、%=、**=。例如，a += b 功能等价于 a = a + b，a -= b 功能等价于 a = a - b，其它复合赋值运算符的功能依此类推。
    - **代码示例：**

```
a = 2
a += 4
print(a)  # 输出：6
b = 6
b -= 2
print(b)  # 输出：4
```

- **逻辑运算符**
    - **定义**：介绍 Python 中分别使用 or、and、not 这三个关键字作为逻辑运算“或”、“与”、“非”的运算符，其中 or 与 and 为双目运算符，not 为单目运算符。
    - **运算符说明：**

        - or 运算符：逻辑或，a or b 当 a 和 b 同时为 False 时结果为 False，否则结果为 True。
        - and 运算符：逻辑与，a and b 当 a 和 b 同时为 True 时结果为 True，否则结果为 False。
        - not 运算符：逻辑非，not a 当 a 为 True 时结果为 False，否则结果为 True。
    - **代码示例：**

```
print(3 > 5 or 4 > 2)  # 输出：True
print(3 > 5 and 4 > 2)  # 输出：False
print(not 3 > 5)  # 输出：True
print(not (3 > 5 and 4 > 2))  # 输出：True
```

- **运算符优先级**
    - **定义**：为了避免含有多个运算符的表达式出现歧义，Python 为每种运算符都设定了优先级。
    - **优先级顺序**：介绍 Python 各种运算符的优先级别从低到高排列如下：
        - or
        - and
        - not
        - in、not in（成员运算符）
        - is、is not（身份运算符）
        - =、+=、-=、*=、/=、//=、%=、**=
        - ==、!=、>、<、>=、<=
        - |、^、&
        - <<、>>
        - +、-（加减号）
        - *、/、//、%
        - +、-（正负号）
        - ~

        - **
    - **代码示例：**

```
print((3 + 4) * 5)  # 输出：35
```

    - **说明：**默认情况下，运算符的优先级决定了复杂表达式中的哪个单一表达式先执行，但用户可以使用圆括号()改变表达式的执行顺序。

8. **常用数学函数**
    - **定义：**介绍 Python 中的一些常用数学函数，这些函数可以用于进行各种数学计算。
    - **常用数学函数说明：**
        - abs(x)：返回数字 x 的绝对值。
        - round(x, ndigits=None)：返回浮点数 x 的四舍五入值，如给出 ndigits 值，则代表舍入到小数点后的 ndigits 位数。
        - math.ceil(x)：返回不小于 x 的最小整数。
        - math.floor(x)：返回不大于 x 的最大整数。
        - math.exp(x)：返回自然常数 e 的 x 次幂。
        - math.log(x)：返回 x 的自然对数。
        - math.log10(x)：返回以 10 为基数的 x 的对数。
        - math.sin(x)、math.cos(x)、math.tan(x)：分别返回 x 弧度的正弦值、余弦值和正切值。
        - math.asin(x)、math.acos(x)、math.atan(x)：分别返回 x 的反正弦弧度值、反余弦弧度值和反正切弧度值。
        - math.degrees(x)：把弧度 x 转换为角度。
        - math.radians(x)：把角度 x 转换为弧度。
    - **代码示例：**

```
print(abs(-10))  # 输出：10
print(abs(2.75))  # 输出：2.75
print(round(8.79342))  # 输出：9
print(round(8.79342, 2))  # 输出：8.79
```

```
import math  # 导入 math 模块
print(math.pi)  # 输出：3.141592653589793
print(math.ceil(7.12))  # 输出：8
print(math.floor(2.83))  # 输出：2
print(math.exp(3))  # 输出：20.085536923187668
print(math.log(100))  # 输出：4.605170185988092
print(math.log10(100))  # 输出：2.0
print(math.sin(math.pi/2))  # 输出：1.0
print(math.asin(1))  # 输出：1.5707963267948966
print(math.degrees(math.asin(1)))  # 输出：90.0
```

（三）课堂总结

1. 回顾数字类型的概念及特点，强调数字类型是不可变数据类型，如果改变数字类型的变量的值，将重新分配内存空间。
2. 总结整型的表示方法，包括二进制、八进制、十进制和十六进制，以及浮点型的表示方法，包括十进制小数形式和科学记数法。
3. 复习复数类型的一般形式及创建方法，以及布尔类型的取值及转换规则。
4. 梳理 Python 中的类型转换方法，包括 int()、float()、bool()等函数的使用，以及类型转换的规则。
5. 复习 Python 中的运算符，包括算术运算符、比较运算符、赋值运算符、逻辑运算符及运算符优先级，以及常用数学函数的使用方法，如 abs()、round()、math.ceil()、math.floor()、math.exp()、math.log()、math.sin()等。

（四）布置作业

第四章 字符串（String）

计划学时：6 学时

一、教学（实践）目标

（一）知识目标

1. 理解字符串的定义、特点（有序字符集合、不可变数据类型）。
2. 掌握字符串的创建方法，包括使用单引号、双引号、三引号以及转义字符、Raw 字符串的使用。
3. 理解并掌握字符串的基本操作，如长度计算、连接、重复连接、成员运算、索引和切片。
4. 掌握字符串常用方法的使用，包括去除两端多余空白、大小写转换、替换、分割、连接、子串查找和计数、判断等。
5. 理解并掌握字符串格式化的三种方法：%符、format()方法和 f-strings。

（二）能力目标

1. 能够熟练进行字符串的创建和基本操作。
2. 能够灵活运用字符串方法解决实际问题，如文本处理、格式转换等。
3. 能够根据需求选择合适的字符串格式化方法，并正确使用。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和解决实际编程问题的能力。
2. 提高学生的自主学习能力，引导学生在学习过程中善于思考、勇于探索。

（四）思政目标

在编程实践中，培养学生注重代码规范和可读性的意识，强调良好的编程习惯对软件质量和维护的重要性，培养学生的责任感和职业素养。

二、教学（实践）重难点

（一）重点

1. 字符串的创建方法，特别是转义字符和 Raw 字符串的使用。
2. 字符串基本操作中的索引和切片，以及它们的规则和应用场景。
3. 字符串常用方法的功能和使用方式，如 strip()、replace()、split()、join()等。
4. 字符串格式化的三种方法的使用，包括它们的格式控制符和格式说明符。

（二）难点

1. 理解字符串切片操作中步长为负数时的取值规则和应用场景。
2. 灵活运用字符串方法解决复杂的文本处理问题，如多步操作的组合使用。
3. 掌握字符串格式化方法中各种格式控制符和格式说明符的使用，以及它们之间的区别和联系。

三、教学（实践）方法

1. **讲授法：**讲解字符串的概念、创建方法、基本操作、常用方法和格式化方法，使学生建立起系统的知识框架。
2. **案例分析法：**通过展示字符串处理的实际案例，如文本提取、格式转换等，帮助学生理解如何将理论知识应用于实际编程场景，加深对知识的理解和掌握。
3. **实践操作法：**安排学生进行编程实践，让学生在实践中熟悉字符串的各种操作和方法的使用，提高学生的动手能力和解决实际问题的能力。

四、教学过程

（一）课前思政、素质元素导入

在当今数字化时代，编程已成为一种重要的技能。良好的编程习惯不仅能够提高代码的可读性和可维护性，还能减少错误的发生。字符串作为编程中常用的数据类型，其规范的使用和高效的处理能力对于软件开发至关重要。引导学生思考如何在编程中注重代码规范，培养良好的编程习惯，为将来从事软件开发工作打下坚实的基础。

（二）教学内容

1. 字符串的创建

- **定义与表示方法：**讲解字符串是有序的字符集合，不可变数据类型。介绍使用单引号、双引号、三引号定义字符串的方法，以及转义字符和 Raw 字符串的使用。通过示例代码展示不同表示方法的字符串创建方式。
- **练习：**让学生尝试创建包含特殊字符的字符串，以及使用 Raw 字符串创建包含路径等信息的字符串。

2. 字符串基本操作

- **长度计算**：介绍 len(s)函数用于计算字符串 s 的长度。通过示例代码展示其使用方法。
- **连接与重复连接**：讲解使用+运算符连接两个字符串，使用*运算符重复连接字符串的方法。通过示例代码展示其效果。
- **成员运算**：介绍使用 in 和 not in 判断字符串是否属于另一个字符串，返回布尔值。通过示例代码展示其使用场景。
- **索引和切片**：
    - **索引**：讲解字符串可以通过索引访问对应位置的值，索引从 0 开始，支持负索引。通过示例代码展示正向索引和负索引的使用，以及索引值超出范围时引发的 IndexError 异常。
    - **切片**：讲解切片操作 var[start:end:step]的含义和规则，包括省略 start、end、step 的情况，以及步长为负数时的取值规则。通过示例代码展示不同切片操作的效果，以及切片操作不会引发异常的特点。
- **练习**：让学生进行字符串长度计算、连接、重复连接、成员运算、索引和切片的操作练习，巩固所学知识。

3. 字符串方法

- **去除两端多余空白**：介绍 s.strip(chars=None)、s.lstrip(chars=None)、s.rstrip(chars=None)方法的功能和使用，以及 chars 参数的默认值和指定字符的使用。通过示例代码展示其效果。
- **大小写转换**：介绍 s.upper()、s.lower()、s.capitalize()、s.swapcase()方法的功能和使用。通过示例代码展示不同大小写转换方法的效果。
- **字符串替换**：介绍 s.replace(old, new, count=-1)方法的功能和使用，包括 count 参数的含义和使用。通过示例代码展示替换操作的效果。
- **字符串分割**：介绍 s.split(sep=None, maxsplit=-1)方法的功能和使用，包括 sep 参数的默认值、指定分隔符和 maxsplit 参数的使用。通过示例代码展示分割操作的效果。
- **字符串连接**：介绍 s.join(str_sequence)方法的功能和使用，通过示例代码展示如何将字符串序列连接成一个新字符串。

- **子串查找**：介绍 s.find(sub[, start[, end]])和 s.index(sub[, start[, end]])方法的功能和使用，以及它们的区别。通过示例代码展示查找子串的操作和 ValueError 异常的引发。
- **子串计数**：介绍 s.count(sub[, start[, end]])方法的功能和使用。通过示例代码展示计数操作的效果。
- **字符串判断**：介绍 s.isalpha()、s.isupper()、s.islower()、s.isdigit()、s.isalnum()、s.isspace()、s.startswith(prefix)、s.endswith(suffix)方法的功能和使用。通过示例代码展示不同判断方法的效果。
- **练习**：让学生使用字符串方法进行去除空白、大小写转换、替换、分割、连接、查找子串、计数字串和判断等操作的练习，加深对字符串方法的理解和掌握。

4. 字符串格式化

- **%符格式化字符串**：
  - 讲解%符格式化字符串的基本形式和格式控制符的含义，包括 typecode（如 s、d、f 等）、flags（如-、+、空格、0 等）、width、.precision 等。
  - 通过示例代码展示不同格式控制符和可选项的使用效果，如左对齐、右对齐、占位宽度、小数点后保留位数等。
  - 讲解使用格式控制符的(name)选项时，如何通过字典传递真实值。
- **format()方法**：
  - 讲解 format()方法的基本形式和格式说明符的含义，包括 index 选项（作为索引或键）、format_spec（如 fill、align、sign、width、.precision、type 等）。
  - 通过示例代码展示 format()方法的使用，包括不使用选项、使用 index 选项作为索引或键、不同格式规格的使用效果。
- **f-strings**：
  - 讲解 f-strings 的基本形式和特点，包括以 f 或 F 引领字符串、使用{}标明替换字段、支持表达式和函数调用。

- 通过示例代码展示 f-strings 的使用，包括变量替换、表达式计算、格式规格的使用，以及引号的使用规则。

- **练习：**让学生使用%符、format()方法和 f-strings 进行字符串格式化的练习，掌握不同格式化方法的特点和使用场景。

（三）课堂总结

1. 回顾本章学习的主要内容，包括字符串的创建、基本操作、常用方法和格式化方法。
2. 强调字符串作为不可变数据类型的特点，以及索引和切片操作的规则。
3. 总结字符串方法的功能和使用场景，以及字符串格式化方法的优缺点和适用情况。
4. 鼓励学生在课后多进行编程实践，熟练掌握字符串的各种操作和方法的使用。

（四）布置作业

第五章 条件判断语句

计划学时：3 学时

一、教学（实践）目标

（一）知识目标

1. 理解条件判断语句的作用和重要性。
2. 掌握 if 语句的基本形式、执行流程以及缩进规则。
3. 理解并掌握 if...elif...else 结构的使用方法和判断流程。
4. 掌握嵌套 if 语句的结构和应用场景。
5. 理解并掌握 if...else 三元表达式的使用方法和特点。

（二）能力目标

1. 能够根据实际问题的需求，正确使用 if 语句进行条件判断。
2. 能够灵活运用 if...elif...else 结构处理多条件判断问题。
3. 能够设计并实现嵌套 if 语句的逻辑结构，解决复杂问题。
4. 能够熟练使用 if...else 三元表达式进行简洁的条件赋值或表达。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和问题分析能力。
2. 提高学生的编程实践能力，引导学生在实践中探索和创新。

（四）思政目标

通过编程实践，培养学生的耐心和专注力，强调在编程中注重逻辑性和规范性的重要性，培养学生的责任感和职业素养。

二、教学（实践）重难点

（一）重点

1. if 语句的基本形式和执行流程。
2. if...elif...else 结构的使用方法和判断逻辑。
3. 嵌套 if 语句的结构和应用场景。
4. if...else 三元表达式的使用方法和特点。

（二）难点

1. 理解嵌套 if 语句的逻辑结构和执行顺序。
2. 灵活运用 if...elif...else 结构处理复杂的多条件判断问题。

3. 掌握 if...else 三元表达式的应用场景和使用技巧。

三、教学（实践）方法

1. **讲授法**：讲解条件判断语句的概念、基本形式、执行流程和应用场景，使学生建立起系统的知识框架。
2. **案例分析法**：通过展示实际编程案例，如判断闰年、成绩等级评定等，帮助学生理解如何将条件判断语句应用于实际问题，加深对知识的理解和掌握。
3. **实践操作法**：安排学生进行编程实践，让学生在实践中熟悉条件判断语句的使用方法，提高学生的动手能力和解决实际问题的能力。

四、教学过程

（一）课前思政、素质元素导入

在编程中，条件判断语句是实现程序逻辑控制的基础，它能够根据不同的条件执行不同的操作。在实际编程中，合理的条件判断逻辑不仅能够提高程序的效率，还能增强程序的可读性和可维护性。引导学生思考如何在编程中注重逻辑性和规范性，培养良好的编程习惯，为将来从事软件开发工作打下坚实的基础。

（二）教学内容

1. if 语句

- **基本形式**：讲解 if 语句的语法结构，包括条件表达式和代码块的组成，以及缩进规则。通过示例代码展示 if 语句的执行流程。
- **执行流程**：强调条件表达式的计算结果决定了代码块是否执行。通过示例代码展示条件为 True 和 False 时的不同执行结果。
- **练习**：让学生编写一个简单的程序，判断一个数是否为正数，并输出相应的提示信息。

2. if...elif...else 结构

- **基本形式**：讲解 if...elif...else 结构的语法和执行流程，包括多个条件表达式的判断顺序和逻辑。通过示例代码展示如何使用 if...elif...else 结构处理多条件判断问题。

- **执行流程**：强调从上到下的判断顺序，以及一旦某个条件为 True，后续条件将被忽略。通过示例代码展示不同条件满足时的执行结果。
- **练习**：让学生编写一个程序，根据输入的成绩输出相应的等级（A、B、C、D、E）。

3. 嵌套 if 语句

- **基本形式**：讲解嵌套 if 语句的概念和结构，即在一个 if 语句内部嵌套另一个 if 语句。通过示例代码展示嵌套 if 语句的使用场景和逻辑结构。
- **执行流程**：强调嵌套 if 语句的执行顺序和逻辑关系，以及如何通过嵌套实现复杂的条件判断。通过示例代码展示嵌套 if 语句的执行过程。
- **练习**：让学生编写一个程序，判断输入的年份是否为闰年。

4. if...else 三元表达式

- **基本形式**：讲解 if...else 三元表达式的语法结构和使用方法，包括表达式 1、条件表达式和表达式 2 的组成。通过示例代码展示 if...else 三元表达式的使用场景和效果。
- **执行流程**：强调条件表达式的结果决定了返回表达式 1 还是表达式 2 的值。通过示例代码展示不同条件下的结果。
- **练习**：让学生使用 if...else 三元表达式完成一个简单的条件赋值操作，如比较两个数的大小并赋值给第三个变量。

（三）课堂总结

1. 回顾本章学习的主要内容，包括 if 语句、if...elif...else 结构、嵌套 if 语句和 if...else 三元表达式的使用方法和应用场景。
2. 强调条件判断语句在编程中的重要性，以及合理使用条件判断逻辑对程序效率和可读性的影响。
3. 鼓励学生在课后多进行编程实践，熟练掌握条件判断语句的使用方法，提高编程能力。

（四）布置作业

第六章 循环语句

计划学时：6 学时

一、教学（实践）目标

（一）知识目标

1. 理解循环语句的作用和重要性。
2. 掌握 while 循环的基本形式、执行流程和应用场景。
3. 掌握 for 循环的基本形式、执行流程和应用场景。
4. 理解循环嵌套的概念和使用方法。
5. 掌握 break 和 continue 语句的功能和使用场景。
6. 理解循环语句中 else 代码块的作用和使用方法。

（二）能力目标

1. 能够根据实际问题的需求，正确使用 while 和 for 循环实现重复操作。
2. 能够灵活运用循环嵌套解决复杂问题。
3. 能够合理使用 break 和 continue 语句控制循环流程。
4. 能够正确使用循环语句中的 else 代码块实现特定逻辑。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和问题分析能力。
2. 提高学生的编程实践能力，引导学生在实践中探索和创新。

（四）思政目标

通过编程实践，培养学生的耐心和专注力，强调在编程中注重逻辑性和规范性的重要性，培养学生的责任感和职业素养。

二、教学（实践）重难点

（一）重点

1. while 循环和 for 循环的基本形式和执行流程。
2. 循环嵌套的结构和应用场景。
3. break 和 continue 语句的功能和使用场景。
4. 循环语句中 else 代码块的作用和使用方法。

（二）难点

1. 理解循环嵌套的逻辑结构和执行顺序。

2. 灵活运用 break 和 continue 语句控制复杂的循环流程。
3. 掌握循环语句中 else 代码块的执行逻辑和应用场景。

三、教学（实践）方法

1. **讲授法**：讲解循环语句的概念、基本形式、执行流程和应用场景，使学生建立起系统的知识框架。
2. **案例分析法**：通过展示实际编程案例，如计算累加和、输入验证等，帮助学生理解如何将循环语句应用于实际问题，加深对知识的理解和掌握。
3. **实践操作法**：安排学生进行编程实践，让学生在实践中熟悉循环语句的使用方法，提高学生的动手能力和解决实际问题的能力。

四、教学过程

（一）课前思政、素质元素导入

在编程中，循环语句是实现程序自动化和高效处理数据的重要工具。合理的循环逻辑不仅能够提高程序的效率，还能增强程序的可读性和可维护性。引导学生思考如何在编程中注重逻辑性和规范性，培养良好的编程习惯，为将来从事软件开发工作打下坚实的基础。

（二）教学内容

1. while 循环

- **基本形式**：讲解 while 循环的语法结构，包括条件表达式和代码块的组成。通过示例代码展示 while 循环的执行流程。
- **执行流程**：强调条件表达式的计算结果决定了代码块是否执行，以及循环的终止条件。通过示例代码展示条件为 True 和 False 时的不同执行结果。
- **练习**：让学生编写一个简单的程序，使用 while 循环计算 1 到 100 的整数的总和。

2. for 循环

- **基本形式**：讲解 for 循环的语法结构，包括临时变量和可迭代对象的组成。通过示例代码展示 for 循环的执行流程。

- **执行流程**：强调 for 循环的执行次数由可迭代对象的元素个数决定，以及如何逐个提取元素执行代码块。通过示例代码展示 for 循环遍历字符串、列表等可迭代对象的效果。
- **练习**：让学生编写一个程序，使用 for 循环遍历一个字符串，并输出每个字符。

3. 循环嵌套

- **基本形式**：讲解循环嵌套的概念，即在一个循环体内部嵌套另一个循环。通过示例代码展示嵌套循环的结构和应用场景。
- **执行流程**：强调嵌套循环的执行顺序和逻辑关系，以及如何通过嵌套实现复杂的重复操作。通过示例代码展示嵌套循环输出九九乘法表的过程。
- **练习**：让学生编写一个程序，使用嵌套循环输出九九乘法表。

4. break 和 continue 语句

- **break 语句**：讲解 break 语句的功能，即跳出整个循环，终止循环的执行。通过示例代码展示 break 语句的使用场景和效果。
- **continue 语句**：讲解 continue 语句的功能，即跳过当前循环的剩余语句，继续执行下一轮循环。通过示例代码展示 continue 语句的使用场景和效果。
- **练习**：让学生编写一个程序，使用 break 和 continue 语句控制循环流程，实现特定的逻辑。

5. 循环语句中的 else 代码块

- **基本形式**：讲解循环语句中 else 代码块的语法结构和作用，即在循环正常结束时执行 else 代码块，除非循环被 break 语句终止。通过示例代码展示 else 代码块的使用方法和效果。
- **执行流程**：强调 else 代码块的执行逻辑，以及如何与 break 语句配合使用实现特定逻辑。通过示例代码展示 else 代码块在循环中的应用场景。
- **练习**：让学生编写一个程序，使用循环语句中的 else 代码块实现特定的逻辑，如判断一个数是否为质数。

（三）课堂总结

1. 回顾本章学习的主要内容，包括 while 循环、for 循环、循环嵌套、break 和 continue 语句、循环语句中的 else 代码块的使用方法和应用场景。
2. 强调循环语句在编程中的重要性，以及合理使用循环逻辑对程序效率和可读性的影响。
3. 鼓励学生在课后多进行编程实践，熟练掌握循环语句的使用方法，提高编程能力。

（四）布置作业

第七章 列表（List）和元组（Tuple）

计划学时：6 学时

一、教学（实践）目标

（一）知识目标

1. 理解列表和元组的概念、特点及区别。
2. 掌握列表和元组的创建方法。
3. 掌握列表和元组的基本操作，包括长度计算、加法、乘法、成员运算、索引和切片。
4. 掌握列表的常用方法，如添加、删除、修改元素，以及排序、复制等。
5. 掌握元组的常用方法，如统计元素出现次数、查找元素索引等。

（二）能力目标

1. 能够根据实际需求选择合适的数据类型（列表或元组）进行数据存储和操作。
2. 能够熟练使用列表和元组的基本操作和方法解决实际问题。
3. 能够灵活运用列表和元组进行数据处理和程序设计。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和数据处理能力。
2. 提高学生的自主学习能力和编程实践能力，引导学生在学习过程中善于思考、勇于探索。

（四）思政目标

通过编程实践，培养学生的耐心和专注力，强调在编程中注重逻辑性和规范性的重要性，培养学生的责任感和职业素养。

二、教学（实践）重难点

（一）重点

1. 列表和元组的创建方法。
2. 列表和元组的基本操作，包括长度计算、加法、乘法、成员运算、索引和切片。
3. 列表的常用方法，如添加、删除、修改元素，以及排序、复制等。
4. 元组的常用方法，如统计元素出现次数、查找元素索引等。

（二）难点

1. 理解列表和元组的区别，以及在实际应用中的选择。
2. 灵活运用列表的切片操作和方法进行数据处理。
3. 掌握元组的不可变性及其在实际应用中的优势。

三、教学（实践）方法

1. **讲授法**：讲解列表和元组的概念、特点、创建方法、基本操作和常用方法，使学生建立起系统的知识框架。
2. **案例分析法**：通过展示实际编程案例，如数据处理、元素查找等，帮助学生理解如何将列表和元组应用于实际问题，加深对知识的理解和掌握。
3. **实践操作法**：安排学生进行编程实践，让学生在实践中熟悉列表和元组的使用方法，提高学生的动手能力和解决实际问题的能力。

四、教学过程

（一）课前思政、素质元素导入

在编程中，列表和元组是常用的数据类型，它们能够有效地组织和处理数据。合理使用列表和元组不仅能够提高程序的效率，还能增强程序的可读性和可维护性。引导学生思考如何在编程中注重逻辑性和规范性，培养良好的编程习惯，为将来从事软件开发工作打下坚实的基础。

（二）教学内容

1. 列表和元组的概念

- **列表**：列表是可变序列，可以随时增加、修改或删除元素。
- **元组**：元组是不可变序列，创建后无法修改。
- **区别**：列表是可变的，元组是不可变的。

2. 列表的创建

- **使用方括号**：`list_1 = [1, 2.0, 'hello']`
- **使用 list()函数**：`list_2 = list('python')`
- **空列表**：`list_3 = []` 或 `list_4 = list()`

3. 列表的基本操作

- **长度计算**：`len(list_1)` 返回列表的长度。
- **加法**：`list_1 + list_2` 将两个列表连接。

- **乘法**：list_1 * 3 将列表重复三次。
- **成员运算**：10 in list_1 判断元素是否在列表中。
- **索引和切片**：
  - **索引**：list_1[0] 访问列表的第一个元素。
  - **切片**：list_1[1:3] 获取列表的第 2 个到第 3 个元素（不包括第 4 个）。

4. 列表的常用方法

- **添加元素**：
  - append(ob)：在列表末尾添加一个元素。
  - extend(iterable)：在列表末尾一次性添加多个元素。
  - insert(idx, ob)：在指定位置插入一个元素。
- **删除元素**：
  - del list_1[0]：删除指定位置的元素。
  - remove(ob)：删除第一个出现的元素。
  - pop(idx)：删除并返回指定位置的元素。
  - clear()：清空列表。
- **修改元素**：通过索引或切片修改列表中的元素。
- **排序**：
  - sort()：对列表进行原位排序。
  - sorted(list)：返回排序后的新列表，原列表不变。
- **反转**：reverse() 将列表中的元素反向排列。
- **统计和查找**：
  - count(ob)：统计元素出现的次数。
  - index(ob)：查找元素的索引位置。

5. 元组的创建

- **使用圆括号**：t1 = (10, 11, 12, 13, 'a', 'b', 'c')
- **单元素元组**：t2 = (1,)
- **使用 tuple()函数**：t3 = tuple(range(10))
- **空元组**：t4 = () 或 t5 = tuple()

6. 元组的基本操作

- **长度计算**：len(t1) 返回元组的长度。
- **加法**：t1 + t2 将两个元组连接。
- **乘法**：t1 * 3 将元组重复三次。
- **成员运算**：10 in t1 判断元素是否在元组中。
- **索引和切片**：
  - **索引**：t1[0] 访问元组的第一个元素。
  - **切片**：t1[1:3] 获取元组的第 2 个到第 3 个元素（不包括第 4 个）。
- **统计和查找**：
  - count(ob)：统计元素出现的次数。
  - index(ob)：查找元素的索引位置。

（三）课堂总结

1. 回顾本章学习的主要内容，包括列表和元组的概念、创建方法、基本操作和常用方法。
2. 强调列表和元组的区别，以及在实际应用中的选择。
3. 鼓励学生在课后多进行编程实践，熟练掌握列表和元组的使用方法，提高编程能力。

（四）布置作业

第八章 字典（Dictionary）

计划学时：3 学时

一、教学（实践）目标

（一）知识目标

1. 理解字典的基本概念、特点及应用场景。
2. 掌握字典的创建方法。
3. 掌握字典的基本操作，包括元素个数、键值访问、添加、修改、删除等。
4. 掌握字典的常用方法，如 get()、setdefault()、update()、pop()、popitem()等。
5. 掌握字典的遍历方法，包括遍历键、值和键值对。

（二）能力目标

1. 能够根据实际需求创建和操作字典。
2. 能够灵活运用字典的方法解决实际问题。
3. 能够通过字典的遍历方法处理数据。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和数据处理能力。
2. 提高学生的自主学习能力和编程实践能力。

（四）思政目标

通过编程实践，培养学生的耐心和专注力，强调在编程中注重逻辑性和规范性的重要性。

二、教学（实践）重难点

（一）重点

1. 字典的创建方法。
2. 字典的基本操作和常用方法。
3. 字典的遍历方法。

（二）难点

1. 理解字典的键必须是不可变数据类型。
2. 灵活运用字典的方法解决实际问题。
3. 掌握字典的遍历方法及其应用场景。

三、教学（实践）方法

1. **讲授法**：讲解字典的概念、创建方法、操作和方法。
2. **案例分析法**：通过实际案例帮助学生理解字典的应用。
3. **实践操作法**：安排编程实践，提高学生的动手能力。

四、教学过程

（一）课前思政、素质元素导入

简述字典在编程中的重要性，强调其在数据存储和处理中的作用，引导学生思考如何在编程中注重逻辑性和规范性。

（二）教学内容

1. 字典的概念

- 字典是由键值对组成的无序可变序列。
- 键必须是不可变数据类型，值可以是任何数据类型。

2. 字典的创建

- 使用花括号：`dict1 = {'Name': 'Tom', 'Age': 27}`
- 使用 `dict()`函数：`dict2 = dict(Name='Tom', Age=27)`
- 使用 `fromkeys()`方法：`dict3 = {}.fromkeys('abc', 1)`

3. 字典的基本操作

- **元素个数**：`len(dict1)`
- **判断键是否存在**：`'Name' in dict1`
- **访问值**：`dict1['Name']`
- **添加和修改元素**：`dict1['Gender'] = 'Male'`
- **删除元素**：
  - `del dict1['Age']`
  - `dict1.pop('Age')`
  - `dict1.popitem()`
- **清空字典**：`dict1.clear()`

4. 字典的常用方法

- **get()**：`dict1.get('Name', 'Default')`
- **setdefault()**：`dict1.setdefault('Hobby', 'Reading')`

- **update()**：dict1.update({'Age': 28, 'Gender': 'Female'})

5. 字典的遍历

- **遍历键**：for key in dict1.keys():
- **遍历值**：for value in dict1.values():
- **遍历键值对**：for key, value in dict1.items():

（三）课堂总结

1. 回顾字典的概念、创建方法、基本操作和常用方法。
2. 强调字典的键必须是不可变数据类型。
3. 鼓励学生在课后多进行编程实践，熟练掌握字典的使用方法。

（四）布置作业

第九章 集合（Set）

计划学时：3 学时

一、教学（实践）目标

（一）知识目标

1. 理解集合的概念、特点及应用场景。
2. 掌握集合的创建方法。
3. 掌握集合的基本操作，包括元素个数、添加、删除、遍历等。
4. 掌握集合的常用方法，如 add()、remove()、discard()、pop()、clear()等。
5. 掌握集合的集合操作，如并集、交集、差集、对称差集等。

（二）能力目标

1. 能够根据实际需求创建和操作集合。
2. 能够灵活运用集合的方法解决实际问题。
3. 能够通过集合的集合操作处理数据。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和数据处理能力。
2. 提高学生的自主学习能力和编程实践能力。

（四）思政目标

通过编程实践，培养学生的耐心和专注力，强调在编程中注重逻辑性和规范性的重要性。

二、教学（实践）重难点

（一）重点

1. 集合的创建方法。
2. 集合的基本操作和常用方法。
3. 集合的集合操作（并集、交集、差集、对称差集）。

（二）难点

1. 理解集合的不可变性和唯一性。
2. 灵活运用集合的集合操作解决实际问题。

三、教学（实践）方法

1. **讲授法**：讲解集合的概念、创建方法、操作和方法。
2. **案例分析法**：通过实际案例帮助学生理解集合的应用。
3. **实践操作法**：安排编程实践，提高学生的动手能力。

四、教学过程

（一）课前思政、素质元素导入

简述集合在编程中的重要性，强调其在数据处理中的作用，引导学生思考如何在编程中注重逻辑性和规范性。

（二）教学内容

1. 集合的概念

- 集合是一个无序的不重复元素序列。
- 集合中的元素必须是不可变的（如整数、浮点数、字符串、元组）。

2. 集合的创建

- 使用花括号：`set1 = {1, 2, 3}`
- 使用 `set()`函数：`set2 = set([1, 2, 3, 1])`（自动去除重复元素）
- 创建空集合：`set3 = set()`

3. 集合的基本操作

- **元素个数**：`len(set1)`
- **判断元素是否存在**：`1 in set1`
- **遍历集合**：`for i in set1: print(i)`
- **添加元素**：
  - `add()`：`set1.add(4)`
  - `update()`：`set1.update({4, 5})`
- **删除元素**：
  - `remove()`：`set1.remove(1)`（如果元素不存在会引发 KeyError）
  - `discard()`：`set1.discard(1)`（如果元素不存在不会引发异常）
  - `pop()`：`set1.pop()`（随机删除一个元素，如果集合为空会引发 KeyError）
  - `clear()`：`set1.clear()`（清空集合）

4. 集合的集合操作

- **并集：**
  - union()：set1.union(set2) 或 set1 | set2
- **交集：**
  - intersection()：set1.intersection(set2) 或 set1 & set2
  - intersection_update()：set1.intersection_update(set2)（更新 set1 为交集）
- **差集：**
  - difference()：set1.difference(set2) 或 set1 - set2
  - difference_update()：set1.difference_update(set2)（更新 set1 为差集）
- **对称差集：**
  - symmetric_difference()：set1.symmetric_difference(set2) 或 set1 ^ set2
  - symmetric_difference_update()：set1.symmetric_difference_update(set2)（更新 set1 为对称差集）

5. 集合的包含关系

- **判断子集：**set1.issubset(set2) 或 set1 <= set2
- **判断真子集：**set1 < set2
- **判断超集：**set1.issuperset(set2) 或 set1 >= set2

（三）课堂总结

1. 回顾集合的概念、创建方法、基本操作和常用方法。
2. 强调集合的不可变性和唯一性。
3. 鼓励学生在课后多进行编程实践，熟练掌握集合的使用方法。

（四）布置作业

第十章 函数（Function）

计划学时：6 学时

一、教学（实践）目标

（一）知识目标

1. 理解函数的概念、特点及应用场景。
2. 掌握函数的定义和调用方法。
3. 掌握函数的参数类型，包括位置参数、关键字参数、默认值参数、不定长参数。
4. 掌握函数的返回值机制。
5. 理解变量作用域，包括局部变量和全局变量。
6. 掌握递归函数的定义和使用。
7. 掌握匿名函数的定义和使用。
8. 掌握 map()、filter()和 sorted()函数的使用。

（二）能力目标

1. 能够根据实际需求定义和调用函数。
2. 能够灵活运用函数的参数类型解决实际问题。
3. 能够正确使用函数的返回值。
4. 能够理解并运用变量作用域。
5. 能够设计和使用递归函数解决复杂问题。
6. 能够使用匿名函数简化代码。
7. 能够使用 map()、filter()和 sorted()函数处理数据。

（三）素质目标

1. 培养学生严谨的逻辑思维能力和代码组织能力。
2. 提高学生的自主学习能力和编程实践能力。

（四）思政目标

通过编程实践，培养学生的耐心和专注力，强调在编程中注重逻辑性和规范性的重要性。

二、教学（实践）重难点

（一）重点

1. 函数的定义和调用。
2. 函数的参数类型及其使用。
3. 函数的返回值机制。
4. 变量作用域。
5. 递归函数的定义和使用。
6. 匿名函数的定义和使用。
7. map()、filter()和 sorted()函数的使用。

（二）难点

1. 理解函数的参数传递机制，特别是位置参数和关键字参数的混合使用。
2. 理解变量作用域及其在函数中的应用。
3. 设计和使用递归函数解决复杂问题。
4. 灵活运用 map()、filter()和 sorted()函数处理数据。

三、教学（实践）方法

1. **讲授法**：讲解函数的概念、定义、调用、参数类型、返回值机制、变量作用域、递归函数、匿名函数以及 map()、filter()和 sorted()函数。
2. **案例分析法**：通过实际案例帮助学生理解函数的应用。
3. **实践操作法**：安排编程实践，提高学生的动手能力。

四、教学过程

（一）课前思政、素质元素导入

简述函数在编程中的重要性，强调其在代码组织和复用中的作用，引导学生思考如何在编程中注重逻辑性和规范性。

（二）教学内容

1. 函数的概念

- 函数是一组实现某种功能的语句集合，可以被重复调用。
- 函数可以提高代码的复用率、降低代码冗余、使程序结构更清晰。

2. 函数的定义和调用

- **定义：**
  - 语法：def 函数名([形式参数表]):［函数文档字符串］函数体［return［表达式]]

- 示例：

```
def PrintHello():
    """print Hello"""
    print("Hello")
```

- **调用：**
  - 语法：函数名([实际参数表])
  - 示例：

```
PrintHello()
```

3. 函数的参数

- **位置参数**：按位置顺序传递参数。
  - 示例：

```
def subtract(x, y):
    return x - y
print(subtract(2, 3))
```

- **关键字参数**：通过关键字指定参数值。
  - 示例：

```
print(subtract(x=2, y=3))
```

- **默认值参数**：在定义函数时为参数指定默认值。
  - 示例：

```
def quad(x, a=1, b=0, c=0):
    return a*x**2 + b*x + c
print(quad(2.0))
```

- **不定长参数：**
  - **位置不定长参数**：*args，接收所有位置参数并以元组形式保存。
    - 示例：

```
def add_1(x, *args):
    total = x
    for arg in args:
        total += arg
```

```
    return total
print(add_1(1, 2, 3, 4))
```

- **关键字不定长参数**：**kwargs，接收所有关键字参数并以字典形式保存。
  - 示例：

```
def add_2(x, **kwargs):
    total = x
    for value in kwargs.values():
        total += value
    return total
print(add_2(10, y=11, z=12, w=13))
```

4. 函数的返回值

- return [表达式] 语句用于退出函数并返回值。
- 如果没有返回值，默认返回 None。
- 示例：

```
def odd_or_even(x):
    if x % 2 == 1:
        return "odd"
    else:
        return "even"
print(f"11 is {odd_or_even(11)}")
```

5. 变量作用域

- **局部变量**：在函数内定义，只在函数内有效。
- **全局变量**：在函数外定义，可以在程序任何位置访问。
- 示例：

```
global_a = "Hi"
def test_global_1():
    print("Hello" + global_a)
test_global_1()
```

6. 递归函数

- 递归函数是调用自身的函数。
- 需要定义递归公式和边界条件。
- 示例：

```
def factorial(num):
    if num == 1:
        return 1
    else:
        return num * factorial(num - 1)
print(factorial(5))
```

7. 匿名函数

- 使用 lambda 定义匿名函数。
- 示例：

```
area = lambda width, height: width * height
print(area(3, 4))
```

8. map()、filter()和 sorted()函数

- map()：对序列中的每个元素应用函数。
  - 示例：

```
a = [2, 3, 4]
print(list(map(lambda x: x**2, a)))
```

- filter()：过滤序列中的元素。
  - 示例：

```
print(list(filter(lambda x: x % 2 == 1, [1, 4, 6, 9])))
```

- sorted()：对序列进行排序。
  - 示例：

```
str_list = ["Function", "Define", "Parameters", "Local"]
print(sorted(str_list, key=len))
```

（三）课堂总结

1. 回顾函数的概念、定义、调用、参数类型、返回值机制、变量作用域、递归函数、匿名函数以及 map()、filter()和 sorted()函数。
2. 强调函数在代码组织和复用中的重要性。
3. 鼓励学生在课后多进行编程实践，熟练掌握函数的使用方法。

（四）布置作业

第十一章 模块（Module）

计划学时：3 学时

一、教学（实践）目标

（一）知识目标

1. 理解模块的概念、作用及分类。
2. 掌握模块的导入方法，包括 import 和 from ... import 语句的使用。
3. 掌握自定义模块的创建和使用方法。
4. 掌握第三方模块的安装和使用方法，熟悉 pip 工具的常用命令。

（二）能力目标

1. 能够根据实际需求创建和使用自定义模块。
2. 能够熟练使用 pip 安装、查看、升级和卸载第三方模块。
3. 能够正确导入和使用标准库模块及第三方模块。

（三）素质目标

1. 培养学生对模块化编程的理解和应用能力。
2. 提高学生的自主学习能力和问题解决能力，鼓励学生探索更多第三方模块的使用。

（四）思政目标

通过学习模块的使用，培养学生的组织能力和团队协作精神，强调代码复用和维护的重要性。

二、教学（实践）重难点

（一）重点

1. 模块的概念、作用及分类。
2. 模块的导入方法，包括 import 和 from ... import 语句的使用。
3. 自定义模块的创建和使用。
4. 第三方模块的安装和使用，pip 工具的常用命令。

（二）难点

1. 理解模块的搜索路径和如何将自定义模块添加到搜索路径中。
2. 灵活使用 pip 工具处理第三方模块的安装和管理。

三、教学（实践）方法

1. **讲授法**：讲解模块的概念、导入方法、自定义模块的创建和第三方模块的安装。
2. **案例分析法**：通过实际案例展示如何创建和使用自定义模块，以及如何安装和使用第三方模块。
3. **实践操作法**：安排学生进行编程实践，创建自定义模块，安装第三方模块，并在程序中使用它们。

四、教学过程

（一）课前思政、素质元素导入

简述模块化编程在软件开发中的重要性，强调其在提高代码可维护性和复用性方面的作用，引导学生思考如何在编程中注重模块化设计。

（二）教学内容

1. 模块概述

- **定义**：模块是一个包含定义和语句的程序文件（通常是后缀为.py 的文件），可以被其他 Python 程序文件导入使用。
- **作用**：
  - 提高代码的可维护性。
  - 方便代码重复利用。
  - 避免命名冲突。
- **分类**：
  - Python 标准库中的内置模块。
  - 第三方模块。
  - 自定义模块。

2. 模块的导入

- **import 语句**：
  - 导入整个模块，可以使用 as 选项为导入的模块指定别名。
  - 如果一次导入多个模块，模块之间用逗号,分隔。
  - 示例：

```
import math
import random as rd
```

```
print(math.pi)
print(rd.randint(1, 5))
```

- **from ... import 语句：**
  - 导入指定的对象，可以直接使用，不再需要模块名.来指定所属的模块。
  - 示例：

```
from math import ceil
from random import *
print(ceil(1.2))
print(randint(1, 8))
```

3. 自定义模块

- **创建：**每个程序文件都可以作为一个模块存在，文件名即为模块名。
- **使用：**在同目录下的另一个程序里导入并使用该模块。
- **示例：**

```
# MyModule.py
def print_hello():
    print("Hello!")
# 使用自定义模块
import MyModule
MyModule.print_hello()
```

- **模块搜索路径：**如果要导入的自定义模块不在当前目录下，需要将被导入模块所在目录添加到 Python 模块的搜索路径里。
  - 示例：

```
import sys
sys.path.append("D:\\MyPythonModule")
```

4. 安装第三方模块

- **pip 工具：**Python 的包管理工具，安装 Python 时默认自动安装。
- **常用命令：**
  - 升级 pip：

```
python -m pip install --upgrade pip
```

- 查看已安装的第三方包的列表：

```
pip list
```

- 查看已安装的某个第三方包的详细信息：

```
pip show <packagename>
```

- 安装指定的第三方包：

```
pip install <packagename>
```

- 升级指定的第三方包：

```
pip install --upgrade <packagename>
```

- 卸载指定的第三方包：

```
pip uninstall <packagename>
```

（三）课堂总结

1. 回顾模块的概念、作用及分类。
2. 强调模块导入方法的重要性，包括 import 和 from ... import 语句的使用。
3. 总结自定义模块的创建和使用方法。
4. 强调第三方模块的安装和管理方法，特别是 pip 工具的常用命令。

（四）布置作业